Induction of constraint logic programs

Michèle Sebag, Céline Rouveirol, Jean-François Puget

Induction of Constraint Logic Programs

Michèle Sebag[1] and Céline Rouveirol[2] and Jean-François Puget[3]

(1) LMS - URA CNRS 317
Ecole Polytechnique, 91128 Palaiseau Cedex, France
Michele.Sebag@polytechnique.fr

(2) LRI - URA CNRS 410
Université Paris-XI, 91405 Orsay Cedex, France
Celine.Rouveirol@lri.fr

(3) ILOG, 9 avenue de Verdun, 94253 Gentilly Cedex
jfpuget@ilog.fr

Abstract. Inductive Logic Programming (ILP) is concerned with learning hypotheses from examples, where both examples and hypotheses are represented in the Logic Programming (LP) language. The application of ILP to problems involving numerical information has shown the need for basic numerical background knowledge (e.g. relation "less than"). Our thesis is that one should rather choose Constraint Logic Programming (CLP) as the representation language of hypotheses, since CLP contains the extensions of LP developed in the past decade for handling numerical variables.
This paper deals with learning constrained clauses from positive and negative examples expressed as constrained clauses. A first step, termed *small induction*, gives a computational characterization of the solution clauses, which is sufficient to classify further instances of the problem domain. A second step, termed *exhaustive induction*, explicitly constructs all solution clauses. The algorithms we use are presented in detail, their complexity is given, and they are compared with other prominent ILP approaches.

1 Introduction

Inductive Logic Programming (ILP) is concerned with supervised learning from examples, and it can be considered a a subfield of Logic Programming (LP): it uses a subset of the definite clause language (e.g. used in Prolog) sometimes extended with some form of negation, to represent both the examples and the hypotheses to be learned [14].

The application of ILP to problems involving numerical information, such as chemistry [7], has shown the need for handling basic numerical knowledge, e.g. relation `less than`. This has often been met by supplying the learner with some ad hoc declarative knowledge [23]. However, one cannot get rid of the inherent limitations of LP regarding numerical variables: functions are not interpreted, i.e. they act as functors in terms. The consequences for that are detailed in section 2.1. Other possibilities are to use built-in numerical procedures [17], or

to map the ILP problem at hand onto an attribute-value induction problem [8, 2, 26, 27].

This paper investigates a radically different approach in order to handle numerical information correctly, namely the use of Constraint Logic Programming (CLP) instead of LP as representation language. Indeed, CLP has been developed in the past decade as an extension of LP to other computation domains than Herbrand terms, including e.g. sets, strings, integers, floating point numbers, boolean algebras (see [6] for a survey). We are primarily interested here in the numerical extensions.

This paper extends a previous work devoted to learning constrained clauses from positive and negative examples represented as definite clauses [21]. The extension concerns the formalism of examples, which are thereafter represented as constrained clauses as well; this allows a number of negative examples to be represented via a single constrained clause.

This language of examples and hypotheses constitutes a major difference with other ILP learners, e.g. FOIL [17], ML-Smart [1], PROGOL [13] or REGAL [3] to name a few.

An equally important difference is that our approach is rooted in the Version Space framework [11]. More precisely the set of solution clauses Th here consists of *all* hypotheses partially complete (covering at least one example) and consistent (admitting no exceptions) with respect to the examples [19]. This contrasts with other learners retaining a few hypotheses in Th, optimal or quasi-optimal with regards to some numerical criterion such as the quantity of information for FOIL, or the Minimum Description Length for PROGOL.

This paper presents a 2-step approach. A computable characterization of Th is constructed in a first step, termed *small induction*; this characterization is sufficient for classification purposes. The explicit characterization of Th is obtained in a second step, termed *exhaustive induction*, which is much more computationally expensive than small induction. This 2-step approach allows one to check whether the predictive accuracy of the theory is worth undergoing the expensive process of explicit construction. Further, we show that exhaustive induction can be reformulated as an equivalent constraint solving problem; thereby, the burden of inductive search can be delegated to an external tool, purposely designed for combinatorial exploration of continuous domains or finite sets.

The rest of the paper is organized as follows. Next section briefly presents CLP. Then the induction setting is extended from LP to CLP: the notions of completeness and consistency of constrained clauses are defined. Section 4 is devoted to building constrained clauses consistent with a pair of examples. This is used in section 5, to characterize the set of solution clauses via *small induction*. *Exhaustive induction* is described in section 6, and section 7 is devoted to a complexity analysis of both algorithms. We conclude with some comparison with previous works and directions for future research.

2 Constraint Logic Programming

This section describes the formalism of constraint logic programming, for it both subsumes logic programming [5] and handles clauses that would require an additional background knowledge to be discovered in ILP.

2.1 The need for CLP

As said above, functions are not interpreted in LP; they are only treated as functors for Herbrand terms. It follows that an equation such as `X - Y = 0` will never be true in a LP program: as sign '$-$' is not interpreted, the two sides of the equation cannot be unified.

In practice, Prolog systems offer a limited form of interpreted functions, using the `is` programming construct. This construct evaluates a ground term built with numerical constants and arithmetic functors, and returns the corresponding numerical value. However, this evaluation only applies to ground terms. Again, the goal `Z is X - Y` will not succeed unless both `X` and `Y` are instantiated with numerical values. Prolog systems also provide some predicates over numerical constants, e.g. `=<`, which suffer from the same limitations.

Thus, in order to handle numerical variables without extending unification, one must carefully design predicate definitions, and use the interpretation of functions when some ground terms are found. Here is a clever example of such a definition, reported from [23]. The goal is to define the `less_than` predicate. First thing is to handle the ground case:

```
X less_than Y ← number(X), number(Y), !, X =< Y.
X less_than X ← number(X).
```

Then, in order to handle the non ground variables, one must introduce explicitly a way to bind the variables. The approach presented in [23] consists in introducing a predicate `float`, that represents a finite set of numerical constants:

```
float(X) ← number(X), !.
float(X) ← member(X, [0, 0.1, 0.2, 0.4, 0.8, 1.6, 3.2, 6.4]).
```

The definition of the inequality predicate can then be extended as follows:

```
X less_than Y ← float(X), float(Y), X =< Y.
X less_than Y ← float(X), float(Delta), Y is X + Delta.
```

Such a clever intensional definition still depends on (and is limited by) an extensional definition of floating point constants.

2.2 Notations and definitions

The key idea of CLP stems from the observation that unification is an algorithm for solving equality constraints between Herbrand terms. Hence, new computation domains can be added to LP if adequate constraint solvers are provided. An alternative to special purpose definitions of predicates and extensional definition of numerical domains, precisely consists of developing an adequate constraint

solver, that extends deduction through built-in interpretation of numerical constants and constructs. The CLP scheme thus generalizes the LP scheme as equation solving is a special case of constraint solving.

This requires the introduction of an *algebraic* semantics. Of course, our aim is not to present here an exhaustive state of the art in CLP (see [24]), but rather to define the basic CLP notions with respect to the classical LP and ILP terminology [9, 14].

Let $\mathcal{L} = \mathcal{L}_a \cup \mathcal{L}_c$, be a definite clause language without function symbols other than constants, where $\mathcal{L}_a$ (respectively $\mathcal{L}_c$) defines the set of uninterpreted (resp. interpreted) predicate symbols.

Definition 1. *In the following, a* constraint *denotes a literal built on a predicate symbol in $\mathcal{L}_c$. An* atom *denotes a literal built on a predicate symbol in $\mathcal{L}_a$.*

Definition 2. *A* constrained clause *is a clause of the form:*

$$H \leftarrow B_1 \wedge \ldots \wedge B_m \wedge c_1 \wedge \ldots \wedge c_n$$

where $H, B_1, \ldots, B_m$ are atoms and $c_1, \ldots, c_n$ are constraints. In the following, $c_1 \wedge \ldots \wedge c_n$ is referred to as the constraint part *of the constrained clause, and $H \leftarrow B_1 \wedge \ldots \wedge B_m$ as to the* definite part *of the constrained clause.*

A constrained logic program *is a finite set of constrained clauses.*

A constrained goal *is a clause of the form: $\leftarrow B_1 \wedge \ldots \wedge B_m \wedge c_1 \wedge \ldots \wedge c_n$, where $B_1, \ldots, B_m$ are atoms and $c_1, \ldots, c_n$ are constraints.*

2.3 Operational Semantics of CLP language

In LP, an answer to a query G with respect to a logic program P is a substitution σ (expressed as a set of equalities on variables of G) such that $G\sigma$ belongs to the least Herbrand model of P. An answer to a query G with respect to a CLP program P is not a substitution any more, but a set of consistent constraints such that all atoms in G have been resolved. We refer to [24] for a formal definition of the inference rule used in CLP, as this is beyond the scope of this paper.

Definition 3. *An* answer *to a CLP goal G with respect to program P is a conjunction of constraints $c_1 \wedge \ldots \wedge c_n$ such that*

$$P, \mathcal{T} \models (\forall)(c_1 \wedge \ldots \wedge c_n \rightarrow G), \qquad \text{or equivalently} \qquad P \models_{\mathcal{S}} (\forall)(c_1 \wedge \ldots \wedge c_n \rightarrow G)$$

where P is a constraint logic program, $\mathcal{S}$ is a structure, $\mathcal{T}$ is the theory axiomatizing $\mathcal{S}$ and $(\forall)F$ denotes the universal closure of F.

The operational semantics of a CLP language can be defined either in terms of logical consequences or in an algebraic way [25] (see [5] for a detailed discussion). From now on, after [24], we use the only notation $\mathcal{D} \models$, which may be read both as the logical or algebraic version of logical entailment.

Definition 4. *A constraint c is* consistent *(or* satisfiable*) if there exists at least one instantiation of variables of c in $\mathcal{D}$ such that c is true, noted $\mathcal{D} \models (\exists)c$.*

A constraint c is consistent with *a set (i.e. conjunction) of constraints σ if* $\mathcal{D} \models (\exists)(\sigma \wedge c)$.

A (set of) constraint(s) σ is inconsistent *if* $\mathcal{D} \models (\forall)(\neg\sigma)$).

Given two (sets of) constraints σ and σ', σ entails σ', noted $\sigma \prec_c \sigma'$, if $\mathcal{D} \models (\forall)(\sigma \rightarrow \sigma')$.

Example: Let variable X have $\mathbb{R}$ as interpretation domain. Then constraint $(X^2 < 0)$ is unsatisfiable; constraint $(X > 10)$ entails constraint $(X > 5)$.

2.4 Domains of computation

Practically, we require the type of any variable X to be set by a *domain constraint* (equivalent to a *selector* in the Annotated Predicate Calculus terminology [10]). This domain constraint gives the initial domain of instantiation Ω_X of the variable. We restrict ourselves to numerical, hierarchical and nominal variables, where Ω_X respectively is (an interval of) $\mathbb{N}$ or $\mathbb{R}$, a tree, or a (finite or infinite) set.

Domain constraints are of the form $(X \in dom(X))$, where $dom(X)$ denotes a subset of Ω_X. The domain constraints considered throughout this paper are summarized in Table 1.

Type of X	Initial domain Ω_X	Domain constraint $X \in dom(X)$
numerical	(interval of) $\mathbb{R}$ or $\mathbb{N}$	$dom(X)$ interval of $\mathbb{R}$ or $\mathbb{N}$.
hierarchical	tree	$dom(X)$ subtree of Ω_X
nominal	finite or infinite set	$dom(X)$ subset of Ω_X

Table 1: Domains of computation and domain constraints

A binary constraint involves a pair of variables X and Y having same domains of instantiation. The advantage of binary constraints is to allow for compact expressions: $(X = Y)$ replaces page-long expression of the form $(X \in \{red\})$ *and* $(Y \in \{red\})$ *or* $(X \in \{blue\})$ *and* $(Y \in \{blue\})$ *or*... The binary constraints considered in this paper are summarized in Table 2.

Type of X and Y	Binary constraints	
numerical	linear inequality	$(X \geq Y + a), (X \leq Y + b)$
hierarchical	generality	$(X \leq Y)$
nominal	equality and inequality	$(X = Y), (X \neq Y)$

Table 2: Domains of computation and binary constraints

Our constraint language is restricted to conjunctions of domain constraints and binary constraints as above. Two reasons explain our choice: this language is sufficient to deal with most real world problems, and it is supported by complete constraint solvers [4].

3 Induction setting in CLP

This section briefly recalls the basic induction setting and the Disjunctive Version Space approach. The key definitions of inductive learning, namely completeness and consistency, are then extended from LP to CLP.

3.1 Learning setting and Disjunctive Version Space

Let the positive and negative examples of the concept to be learned be expressed in the language of instances $\mathcal{L}_i$, and let $\mathcal{L}_h$ denote the language of hypotheses. Let two boolean relations of coverage and discrimination be defined on $\mathcal{L}_h \times \mathcal{L}_i$, respectively telling whether a given hypothesis covers or discriminates a given example.
The basic solutions of inductive learning consist of hypotheses that are complete (cover the positive examples) and consistent (discriminate the negative examples).

The Version Space (VS) framework gives a nice theoretical characterization of the set of solutions [11].Unfortunately noisy examples and disjunctive target concepts lead VS to fail, which implies that VS is not applicable to real-world problems[1]. The Disjunctive Version Space (DiVS) algorithm overcomes these limitations via relaxing the completeness requirement [19]. More precisely, DiVS constructs the set Th of all hypotheses that are partially complete (cover at least one example) and consistent. This is done by repeatedly characterizing the set $Th(E)$ of consistent hypotheses covering E, for each training example E.

The elementary step of Disjunctive Version Space actually consists of constructing the set $D(E,F)$ of hypotheses covering E and discriminating some other training example F: if $F_1, F_2, \ldots, F_n$ denote the training examples not belonging to the same target concept as E, termed *counter-examples* to E, then by construction

$$Th(E) = D(E,F_1) \wedge \ldots \wedge D(E,F_n)$$

3.2 From ILP to CLP

When the current training example E is a definite clause, we proposed to express E as $C\theta$, where C is the definite clause built from E by turning every occurrence of a term t_i in E into a distinct variable X_j, and θ is the substitution given by $\{X_j/t_i\}$ [18]:

$$E = C\theta$$

This decomposition allows induction to independently explore the lattice of definite clauses generalizing C, and the lattice of substitutions or constraints over the variables in C, that generalize θ: as a matter of fact, a substitution is a particular case of constraint (a set of equality constraints between Herbrand terms).

When training examples are described by constrained clauses, we must first get rid of the fact that one constrained clause may admit several equivalent expressions.

Definition 5. *Let g be a constrained clause. The canonical form of g is defined as $G\gamma$, where*
- *G is the definite clause built from g by deleting the constraints and turning*

[1] Real examples are *always* noisy; real target concepts are usually disjunctive.

every occurrence of a term t_i in g into a distinct variable X_j;
• *γ is the maximally specific conjunction of constraints entailed by the constraint part of g and the constraints $(X_j = t_i)$.*

Example: Let g be a constrained clause describing some poisonous chemical molecules:

$$g : poisonous(X) \leftarrow atm(X, Y, carbon, T), atm(X, U, carbon, W), (Y \neq U), (T > W - 2)$$

The canonical expression of g is $G\gamma$, with

$$G : poisonous(X) \leftarrow atm(X', Y, Z, T), atm(X", U, V, W)$$
$$\gamma : (Y \neq U), (T > W - 2), (X = X'), (X = X"), (X' = X"), (Z = carbon), (V = carbon), (Z = V)$$

In the remainder of this paper, "constrained clause" is intended as "constrained clause in canonical form".

Let $E = C\theta$ hereafter denote the constrained clause to generalize. The language of hypotheses $\mathcal{L}_h$ is that of constrained clauses $G\gamma$ where G is a definite clause generalizing C in the sense of θ-subsumption [14], noted $C \prec G$, and γ is a conjunction of constraints set on variables in C, such that θ entails γ (Definition 4):

$$\mathcal{L}_h = \{G\gamma, \text{ such that } C \prec G \text{ and } \theta \prec_c \gamma\}$$

DiVS thus explores a bound logical space with bottom C, and a bound constraint space with bottom θ.

3.3 Completeness and Consistency in CLP

The generality order on constrained clauses is extended from the generalization order on logical clauses defined by θ-subsumption [14], and from the generalization order defined by constraint entailment [6].

Definition 6. *Let $G\gamma$ and $G'\gamma'$ be constrained clauses; $G\gamma$* generalizes *$G'\gamma'$, noted $G'\gamma' \prec_h G\gamma$, if there exists a substitution σ on G such that $G\sigma$ is included in G', and $\gamma'\sigma$ entails γ:*

$$G'\gamma' \prec_h G\gamma \quad \textit{iff} \quad \textit{there exists } \sigma \,/\, G\sigma \subseteq G' \textit{ and } \gamma'\sigma \prec_c \gamma$$

It follows from Definition 6, that any constrained clause $G\gamma$ in the search space $\mathcal{L}_h$, generalizes E (σ being set to the identity substitution on C):

$$G\gamma \in \mathcal{L}_h \quad \textit{implies} \quad C\theta = E \prec_h G\gamma$$

Positive examples are represented as constrained clauses concluding to the predicate to be learned *tc*. Negative examples are also represented as constrained clauses. Indeed, there is no standard semantics for the negation in Logic Programming, and even less for CLP. We therefore explicitly introduce the negation

of target predicate tc, noted ${}^{opp}tc$; negative examples are constrained clauses concluding to ${}^{opp}tc$. For instance, if `active` is the target predicate, we introduce the opposite predicate symbol ^{opp}`active` (= `inactive`).

Then, for any constrained clause g, let ${}^{opp}g$ be defined as the constrained clause obtained from g by replacing the predicate in the head of g, by the opposite target predicate.

$$ {}^{opp}g \ : \quad {}^{opp}head(g) \leftarrow body(g) $$

The consistency of a constrained clause is defined as follows:

Definition 7. *Let $G\gamma$ and $G'\gamma'$ be constrained clauses. $G\gamma$ is* inconsistent *with respect to $G'\gamma'$ iff there exists a substitution σ on G such that $G\sigma$ is included into ${}^{opp}G'$ and γ is consistent with $\gamma'\sigma$:*

$G\gamma$ is *inconsistent* wrt $G'\gamma'$ iff $\exists\ \sigma$ such that $G\sigma \subseteq {}^{opp}G'$ and $\mathcal{D} \models (\exists)(\gamma \wedge \gamma'\sigma)$

Such a substitution σ is termed negative substitution on G *derived from $G'\gamma'$.*

$G\gamma$ discriminates *$G'\gamma'$, if there exists no negative substitution σ on G derived from $G'\gamma'$.*

Example: Let g and g' be two constrained clauses as follows:

g : $poisonous(X) \leftarrow atm(X, Y, carbon, T), atm(X, U, carbon, W), (T > W - 2)$
g' : ${}^{opp}poisonous(X) \leftarrow atm(X, Y, Z, T), atm(X, U, Z, W), (T \leq W)$

Then, g is inconsistent wrt to g': σ being set to the identity substitution, one sees that a molecule involving two carbon atoms with same valence ($T = W$) would be considered both *poisonous* according to g, and *non poisonous* according to g'.

4 Building discriminant constrained clauses

This section focuses on the elementary step of Disjunctive Version Space, namely constructing the set $D(E, F)$ of constrained clauses covering E and discriminating F (in the sense of definition 7), where E and F are constrained clauses concluding to opposite target concepts. We assume in this section that E is consistent with respect to F.

Given the chosen hypothesis language, there exists two ways for a candidate hypothesis $G\gamma$ to discriminate F: The first one, examined in section 4.1, operates on the definite clause part of $G\gamma$: $G\gamma$ discriminates F if G involves a predicate that does not occur in F. The second one, examined in sections 4.2 and 4.3, operates on the constraint part of $G\gamma$: $G\gamma$ discriminates F if γ is inconsistent with the constraint part of F.

4.1 Discriminant predicates

Due to the fact that $\mathcal{C}$ involves distinct variables only, any clause G subsuming $\mathcal{C}$ discriminates F iff it involves a predicate symbol that does not occur in F, termed *discriminant predicate*. Predicate-based discrimination thereby amounts to boolean discrimination (presence/absence of a predicate symbol). More formally,

Proposition 1. *Let $G_{pred}(F)$ be the set of clauses $head(\mathcal{C}) \leftarrow p_i()$., for p_i ranging over the set of discriminant predicate symbols. Then, a definite clause that subsumes $\mathcal{C}$ discriminates F iff it is subsumed by a clause in $G_{pred}(F)$.*

$G_{pred}(F)$ thereby sets an upper bound on the set of definite clauses that subsume $\mathcal{C}$ and discriminate F. Note this set can be empty: e.g. in the chemistry domain, all example molecules are described via the same predicates (*atom* and *bond*), regardless of their class (*poisonous* or *non poisonous*).

4.2 Discriminant domain constraints

Let G be the generalization of $\mathcal{C}$ obtained by dropping all discriminant predicates. With no loss of generality, F can be described[2] as ${}^{opp}G\rho$, with ρ being the constraint part of F.

Hence, G is inconsistent with F; and due to the fact that $\mathcal{C}$ (and hence G) involves distinct variables only, any negative substitution on G derived from F (Definition 7) is a permutation of variables in G. Let Σ denote the set of these negative substitutions. Note that constraints on G are trivially embedded onto constraints on $\mathcal{C}$.

One is finally interested in the following constraints on $\mathcal{C}$:

- Constraint θ which is the constraint part of example E,
- Constraint ρ which is the constraint part of example F,
- And the set Σ of negative substitutions derived from F (being reminded that substitutions are particular cases of constraints).

Let us first concentrate on domain constraints, and assume in this subsection that our constraint language is restricted to domain constraints[3]. A constraint γ is thus composed of a conjunction of domain constraints $(X_i \in dom_\gamma(X_i))$, for X_i ranging over the variables in $\mathcal{C}$.

It is straightforward to show that the lattice of constraints on $\mathcal{C}$ is equivalent to the lattice $\mathcal{L}_{eq} = \mathcal{P}(\Omega_1) \times \mathcal{P}(\Omega_2) \times \ldots$, where Ω_i denotes the domain of instantiation of X_i, for X_i ranging over the variables of $\mathcal{C}$, and $\mathcal{P}(\Omega_i)$ denotes the power set of Ω_i. An equivalent representation of γ is given by the vector of subsets $dom_\gamma(X_i)$.

[2] The predicates appearing in F and not appearing in E can be dropped with no loss of information: given the hypothesis language, they will not be considered in $D(E, F)$.

[3] This restricted language does not include the substitutions, as it does not allow the representation of variable linking. This will be settled in section 4.3.

Building discriminant domain constraints is thus amenable to attribute-value discrimination: two constraints are inconsistent iff they correspond to non overlapping elements in $\mathcal{L}_{eq}$.

Proposition 2. *Let γ be a conjunction of domain constraints $(X_i \in dom_\gamma(X_i))$, and let $\gamma' = (X_{i_0} \in dom_{\gamma'}(X_{i_0})$ be a domain constraint. Constraint γ' is inconsistent with constraint γ iff $dom_\gamma(X_{i_0})$ and $dom_{\gamma'}(X_{i_0})$ are disjoint.*

Let us now characterize the constraints discriminating example F. By definition, $G\gamma$ discriminates F iff γ is inconsistent with $\rho\sigma$ for all σ in Σ.

Definition 8. *An* elementary discriminant constraint *with respect to a negative substitution σ and a variable X, is a domain constraint on X that is entailed by θ and inconsistent with $\rho\sigma$.*
A maximally general elementary discriminant constraint wrt σ and X is called maximally discriminant.

In the considered domain constraint language (section 2.4), there exists at most one maximally discriminant constraint wrt a negative substitution σ and a variable X, noted $(X \in dom_{\sigma^*}(X))$:

- if X is a numerical variable, such a maximally discriminant constraint exists iff $dom_\theta(X)$ et $dom_\rho(X.\sigma)$ are disjoint, in which case $dom_{\sigma^*}(X)$ is the largest interval including $dom_\theta(X)$ and excluding $dom_\rho(X.\sigma)$.
- if X is a hierarchical variable, such a maximally discriminant constraint exists iff $dom_\theta(X)$ et $dom_\rho(X.\sigma)$ are subtrees which are not comparable, in which case $dom_{\sigma^*}(X)$ is the most general subtree that includes $dom_\theta(X)$ and does not include $dom_\rho(X.\sigma)$.
- if X is a nominal variable, such a maximally discriminant constraint exists iff $dom_\theta(X)$ et $dom_\rho(X.\sigma)$ do not overlap, in which case $dom_{\sigma^*}(X)$ is the complementary in Ω_X of $dom_\rho(X.\sigma)$. For the sake of convenience, domain constraint $(X \in dom_{\sigma^*}(X))$ is noted $(X \notin dom_\rho(X.\sigma))$.

If $dom_{\sigma^*}(X)$ exists, X is said to be *σ-discriminant.*

By construction, a domain constraint on X that is entailed by θ and discriminates $\rho\sigma$ must entail $(X \in dom_{\sigma^*}(X))$. An upper bound on the domain constraints that discriminate $\rho\sigma$ is then given by the disjunction of constraints $(X \in dom_{\sigma^*}(X))$, for X ranging over the σ-discriminant variables in C. More formally,

Proposition 3. *Let $var(C)$ be the set of variables in C, let σ be a substitution in Σ, and let γ_σ be the disjunction of constraints $(X_i \in dom_{\sigma^*}(X_i))$ for X_i ranging over the σ-discriminant variables in $var(C)$. Let γ be a conjunction of domain constraints on variables in C that is entailed by θ. Then,*

$$\gamma \text{ is inconsistent with } \rho\sigma \qquad \text{iff} \qquad \gamma \prec_c \gamma_\sigma$$

Example: Let E and F be as follows:

$E : poisonous(X) \leftarrow atm(X, Y, carbon, T), atm(X, U, carbon, W), T < 24, W \geq 25$
$F : {}^{opp}poisonous(X) \leftarrow atm(X, Y, hydrogen, 18), atm(X, U, carbon, W'), W' \leq 21$

The definite clause C built from E is given below; variables Z and V are nominal, with domain of instantiation $\{carbon, hydrogen, oxygen, \ldots\}$. Variables T and W are continuous, with domain of instantiation $\mathbf{N}$. (Other variables are discarded as they do not convey discriminant information).

$$C : poisonous(X) \leftarrow atm(X', Y, Z, T), atm(X", U, V, W)$$

There is no discriminant predicate ($G = C$); Σ includes four negative substitutions $\sigma_1, \sigma_2, \sigma_3$ and σ_4 which correspond to the four possible mappings of the two literals *atm* in C onto the two literals *atm* in F.
Table 3 shows a tabular representation of the constraints θ and $\rho\sigma_i$, where a case of the matrix is a sub domain of the domain of instantiation of the variable.

	X	X'	Y	Z	T	$X"$	U	V	W
θ	–	–	–	*carbon*	$[0, 24)$	–	–	*carbon*	$[25, \infty)$
$\rho\sigma_1$	–	–	–	*hydrogen*	18	–	–	*carbon*	$[0, 21]$
$\rho\sigma_2$	–	–	–	*carbon*	$[0, 21]$	–	–	*hydrogen*	18
$\rho\sigma_3$	–	–	–	*hydrogen*	18	–	–	*hydrogen*	18
$\rho\sigma_4$	–	–	–	*carbon*	$[0, 21]$	–	–	*carbon*	$[0, 21]$

Table 3: Tabular representation of domain constraints

And the (disjunctive) constraint γ_{σ_1} entailed by θ and maximally general such that it is inconsistent with $\rho\sigma_1$ is given as (with $[W \in (21, \infty)]$ written $[W > 21]$ for the sake of readability):

$$\gamma_{\sigma_1} = [Z \notin \{hydrogen\}] \vee [W > 21]$$

4.3 Discriminant binary constraints

We showed that building discriminant binary constraints is amenable to building discriminant domain constraints, via introducing auxiliary constrained variables, termed *relational variables* [21].

As an example, let us consider binary equality or inequality constraints $X = Y$ or $X \neq Y$. One associates to any pair of variables X and Y having same domain of instantiation, the relational variable $(X^{=}Y)$, interpreted for any substitution σ of C as: $(X^{=}Y).\sigma = true$ if $X.\sigma = Y.\sigma$, $(X^{=}Y).\sigma = false$ if $X.\sigma$ and $Y.\sigma$ are distinct constants, and $(X^{=}Y).\sigma$ is not bound otherwise.

Equality constraint $(X = Y)$ (respectively inequality constraint $(X \neq Y)$) is equivalent to domain constraints on relational variable $(X^{=}Y)$ given as $((X^{=}Y) = true)$ (resp. $((X^{=}Y) = false)$).
Binary arithmetic constraint can similarly be built as domain constraints on relational numerical variables: let $(X^{-}Y)$ be the constrained variable interpreted as the difference of numerical variables X and Y, the domain constraint $((X^{-}Y) \in [a, b])$ is equivalent to the binary constraint on X and Y : $(Y + a \leq X \leq Y + b))$.

In the chosen constraint language, all binary constraints can be expressed as domain constraints on such auxiliary variables. Proposition 3 then generalizes as :

Proposition 4. *Let $var^*(\mathcal{C})$ be the set of initial and relational variables in $\mathcal{C}$, let σ be a negative substitution in Σ, and let γ_σ now denote the disjunction of constraints $(X \in dom_{\sigma^*}(X))$ for X ranging over the σ-discriminant variables in $var^*(\mathcal{C})$. Let γ be a conjunction of domain constraints on variables in $var^*(\mathcal{C})$ that is entailed by θ. Then,*

$$\gamma \text{ is inconsistent with } \rho\sigma \qquad \text{iff} \qquad \gamma \prec_c \gamma_\sigma$$

Constraint γ_σ hence is the upper-bound on the set of constraints on $\mathcal{C}$ that are entailed by θ and are inconsistent with $\rho\sigma$.

As an example, the tabular representation (Table 3) is extended to binary constraints as well:

	X	X'	Y	Z	T	$X"$	U	V	W
θ	–	–	–	*carbon*	$[0,24)$	–	–	*carbon*	$[25,\infty)$
$\rho\sigma_1$	–	–	–	*hydrogen*	18	–	–	*carbon*	$[0,21]$
$\rho\sigma_2$	–	–	–	*carbon*	$[0,21]$	–	–	*hydrogen*	18
$\rho\sigma_3$	–	–	–	*hydrogen*	18	–	–	*hydrogen*	18
$\rho\sigma_4$	–	–	–	*carbon*	$[0,21]$	–	–	*carbon*	$[0,21]$

Table 4: domain constraints and

$Z = U$	$W - T$
false	$[1,\infty)$
false	$[-18,3]$
false	$[-3,18]$
true	0
true	0

binary constraints

And the disjunctive constraint γ_{σ_4} entailed by θ and maximally general such that it is inconsistent with $\rho\sigma_4$ is given as:

$$\gamma_{\sigma_4} = [W > 21] \vee [Z \neq U] \vee [W - T > 0]$$

Last, one considers the conjunction of the constraints γ_σ for σ ranging in Σ:

Proposition 5. *Let G be a generalization of $\mathcal{C}$ inconsistent with respect to F, and let γ_F be the conjunction of constraints γ_σ for σ ranging in Σ. Then $G\gamma$ discriminates F iff γ entails γ_F.*

Constraint γ_F thus defines an upper bound on the constraints discriminating F, like $G_{pred}(F)$ is the upper-bound on the set of definite clauses that generalize $\mathcal{C}$ and discriminate F. These are combined in the next section in order to characterize all consistent partially complete constrained clauses.

5 Small induction

Our goal is here to characterize the Disjunctive Version Space learned from positive and negative constrained clauses, and to use this characterization to classify further instances of the problem domain.

5.1 Characterizing $Th(E)$

Let all notations be as in the previous section, and let $G\gamma$ be a constrained clause in the hypothesis language. By recollecting results in sections 4.1 and 4.3, $G\gamma$ discriminates F iff either G is subsumed by a clause in $G_{pred}(F)$ or γ entails γ_F:

Proposition 6. *Let $D(E,F)$ be the set of constrained clauses that generalize E and discriminate F, and let $G\gamma$ be a constrained clause generalizing E. Then $G\gamma$ belongs to $D(E,F)$ if and only if*

$$(\ \exists G' \text{ in } G_{pred}(F) \text{ s.t. } G \prec G') \qquad \text{or} \qquad (\ \gamma \prec_c \gamma_F\ ) \tag{1}$$

And the set $Th(E)$ of consistent constrained clauses covering E can be characterized from the set of constrained clauses covering E and discriminating F, for F ranging over the counter-examples $F_1, \ldots F_n$ to E (i.e. the training examples concluding to the concept opposite to that of E); by construction,

$$Th(E) = D(E,F_1) \wedge \ldots \wedge D(E,F_n)$$

In other words, the pairs $(G_{pred}(F_i), \gamma_{F_i})$ constitute a computational characterization of $Th(E)$: they give means to check whether any given constrained clause belongs to $Th(E)$.

The Disjunctive Version Space finally is constructed by iteratively characterizing $Th(E)$, for E ranging over the training set.

However, looking for consistent hypotheses make little sense when dealing with real-world, hence noisy, data. One is therefore more likely interested in hypotheses admitting a limited number of inconsistencies. Let $Th_\varepsilon(E)$ denote the set of hypotheses covering E and admitting at most ε inconsistencies. Then, we show that $Th_\varepsilon(E)$ can be characterized from the pairs $(G_{pred}(F_i), \gamma_{F_i})$, with no additional complexity [19]: a constrained clause $G\gamma$ covering E belongs to $Th_\varepsilon(E)$ iff it satisfies condition (1) above, for all but at most ε counter-examples F_i to E.

The advantage of this approach is to delay the choice of the consistency bias, from induction to classification, at no additional cost [19]:
Induction constructs once and for all the pairs $(G_{pred}(F_i), \gamma_{F_i})$, or a tractable approximation of these [22];
This allows one to tune the degree of consistency of the hypotheses used during classification, at no extra cost[4].

5.2 Classification in Disjunctive Version Space

One major result of this approach is that the computational characterization of the Disjunctive Version Space is sufficient to classify any further instance of the problem domain. In other words, the explicit construction of $Th(E)$, for E ranging over the training examples, gives no extra prediction power.

[4] The degree of generality of hypotheses can also be tuned at no extra cost; see [19, 22].

The Disjunctive Version Space includes hypotheses concluding to opposite target concepts, since positive *and* negative examples are generalized. And, though these hypotheses are consistent with the training examples, they usually are inconsistent with one another. Classification therefore does not rely on standard logic, but rather on a nearest-neighbor like approach. The instance I to classify is said to be neighbor of a training example E, if I is generalized by a hypothesis in $Th(E)$; I is thereafter classified in the class of the majority of its neighbors.

One shows that I is generalized by a hypothesis in $Th(E)$ iff it is generalized by a hypothesis in $D(E,F)$, for every counter-example[5] F. And this can be checked from the computational characterization of $D(E,F)$:

Proposition 7. *Let I be an instance of the problem domain, formalized as a conjunction of constrained atoms. Then I is generalized by the body of a clause in $D(E,F)$ iff there exists a generalization G of C and a constraint γ such that the body of $G\gamma$ generalizes I, and either G is subsumed by a clause in $G_{pred}(F)$ or γ entails γ_F.*

The important distinction compared to Prop 6. is that γ is not required to be entailed by θ any more: Prop 7 only requires to consider the substitutions between C and the definite part of I.

5.3 A two-step induction

We thus propose a two step induction scheme. During the first step, called *small induction*, all pairs of training examples (E,F) satisfying opposite target concepts are considered; and for each such pair, we build the set of discriminant definite clauses $G_{pred}(F)$ and the discriminant constraint γ_F (conjunction of disjunctions). As shown above, this is sufficient to address the classification of unseen examples, and characterize the set of consistent partially complete constrained clauses.

During the second step, called *exhaustive induction*, all such consistent constrained clauses are explicitly built, and it is shown in the next section that exhaustive induction can be achieved by constraint solving.

The advantage of this scheme is twofold. First, the burden of explicitly constructing the hypotheses can be delegated to constraint solvers, that is, algorithms external to induction and geared for combinatorial search in discrete and continuous domains.

Second, small induction can be viewed as an on-fly, lazy learning, the complexity of which is much smaller than that of exhaustive induction (section 7): it constructs theories which are not understandable, but yet operational to classify examples. One may then get some idea of the accuracy of a theory, before undergoing the expensive process of making it explicit.

[5] Or for all except ε counter-examples, in case the consistency requirement is relaxed.

6 Exhaustive induction

We present here an algorithm called ICP, for *Inductive Constraint Programming*, that constructs the set $Th(E)$ of consistent constrained clauses covering E. In the line of Version Spaces [11], we limit ourselves to construct explicitly the upper bound of $Th(E)$, i.e. the set $G(E)$ of maximally general constrained clauses in $Th(E)$.

ICP proceeds as follows. It first builds the computational characterization of $Th(E)$, i.e. the set of clauses $G_{pred}(F_i)$ and constraint γ_{F_i} for F_i ranging over the counter-examples of E. $G(E)$ is initialized to the empty set and the current constrained clause $G\gamma$ is initialized to the clause $head(\mathcal{C}) \leftarrow$.
Then, $G_{pred}(F_i)$ and γ_{F_i} are explored in depth-first, and clause $G\gamma$ is specialized until it discriminates all counter-examples F_i. All consistent constrained clauses are obtained by backtracking on the specialization choices.

Build G(E)

Init

For $F = F_1, \ldots, F_n$,
  Build $G_{pred}(F)$ and γ_F.
  * Prune γ_F.
$G(E) = \phi$.
$G = head(\mathcal{C}) \leftarrow$.
$\gamma = True$.

Main Loop

For $F = F_1, \ldots, F_n$,
  if G is not subsumed by any clause in $G_{pred}(F)$
  and γ does not entail γ_F, then
    If possible,
      * Specialize $G\gamma$ to discriminate F
    Otherwise,
      Backtrack on the specialization choices
* If $G\gamma$ is maximally general in $Th(E)$,
  $G(E) = G(E) \bigcup \{G\gamma\}$.
Backtrack on the specialization choices.

Specialize $G\gamma$ to discriminate F (non deterministic)

Select a clause G_0 in $G_{pred}(F)$.
  Do $body(G) = body(G) \wedge body(G_0)$
Or,
For each negative substitution σ_k derived from F
  If γ does not entail γ_{σ_k},
    Select a variable X_j that is σ_k-discriminant
      Do $\gamma = \gamma \wedge (X_j \in dom_{\sigma_k^*}(X_j))$

In this scheme, constraint solving is employed to several tasks (indicated with an asterisk):

It is used to prune γ_F: a partial order noted $<_E$ can be defined on the negative substitutions with respect to the positive substitution [20]. Minimal substitutions with respect to this partial order can be viewed as "near-misses": all substitutions but the minimal ones, can soundly be pruned. This pruning was explicitly dealt with in previous works [18, 20]. It turns out to be a special case of constraint entailment ($\sigma_i <_E \sigma_j$ is equivalent to $\gamma_{\sigma_i} \prec_c \gamma_{\sigma_j}$) and this pruning can therefore be achieved by a constraint solver.

It chiefly allows for building $G\gamma$, through selecting specialization choices, checking whether the current solution $G\gamma$ is subsumed by a clause in $G_{pred}(F_i)$, and backtracking.

Last, it allows for testing whether $G\gamma$ is maximally general[6] in $Th(E)$.

7 Complexity

Assume that the domain of instantiation of any variable can be explored with a bounded cost. Then, the complexity of building the maximally discriminant constraint γ_σ that discriminates a negative substitution σ, is linear in the number of initial and relational variables in $\mathcal{C}$. In our constraint language, this complexity is quadratic in the number $\mathcal{X}$ of variables in $\mathcal{C}$.

If $\mathcal{L}$ denotes an upper bound on the number of negative substitutions derived from a counter-example (the size of Σ), the complexity of building γ_F is then $\mathcal{O}(\mathcal{X}^2 \times \mathcal{L})$. The complexity of building $G_{pred}(F)$ (section 4.1) is negligible compared to that of building γ_F (it is linear in the number of predicate symbols in E, which is upper-bounded by $\mathcal{X}$).

Finally the computational characterization of $D(E, F)$ has complexity $\mathcal{O}(\mathcal{X}^2 \times \mathcal{L})$.

Characterizing the Disjunctive Version Space Th requires all pairs $D(E_i, F_j)$ to be characterized; if N denotes the number of training examples, the computational characterization of Th has complexity $\mathcal{O}(\mathcal{X}^2 \times \mathcal{L} \times N^2)$.

The complexity of classifying an unseen example I from Th (proposition 7) is the size of the implicit characterization of Th times the number of substitutions derived from I, upper bounded by $\mathcal{L}$; the complexity of classification hence is $\mathcal{O}(\mathcal{X}^2 \times \mathcal{L}^2 \times N^2)$.

The complexity of the intentional characterization of Th, via algorithm ICP, is in $\mathcal{O}(N \times (\mathcal{X}^{2 \times \mathcal{L} \times N}))$. Needless to say, the learning and classifying processes based on the computational characterization of Th are much more affordable than those based on the explicit characterization of Th.

The typical complexity of first order logic appears through factor $\mathcal{L}$: if M is an upper bound on the number of literals based on a same predicate symbol that

[6] The fact that $G\gamma$ is a maximally general element of $Th(E)$ can be expressed via considering new constraint programs, involving the assertion of all but one elementary constraints satisfied by γ, and the negation of the remaining one. $G\gamma$ is maximally general if such new constraint programs are satisfiable.

occur in an example, and P is the number of predicate symbols, $\mathcal{L}$ is in $M^{M \times P}$. For instance, in the mutagenesis problem [7], examples are molecules involving up to 40 atoms; $\mathcal{L}$ is then 40^{40}.

We therefore used a specifically devised heuristic to overcome this limitation. The exhaustive exploration of the set Σ of negative substitutions, was replaced by a stochastic exploration: we limit ourselves to consider a limited number η of samples in Σ, extracted by a stochastic sampling mechanism [22]. An approximation of $D(E, F)$ was therefore constructed in polynomial time ($\mathcal{O}(\mathcal{X}^2 \times \eta \times N^2)$; to give an order of idea, the number η of samples considered in Σ was limited to 300 (to be compared to 40^{40}). This approach led to outstanding experimental results, compared to the state of the art on the mutagenesis problem [23].

8 Discussion and Perspectives

This section first discusses our choice of a maximally discriminant induction, then situates this work with respect to some previous works devoted to generalization of constraints [16, 12] or reformulation of ILP problems [8, 26, 27].

8.1 Generalization Choices

This work first extends the frame of induction to constraint logic programming; see [22] for an experimental demonstration of the potentialities of this language. Note that this frame does *not allow* to learn clauses that could not be learned by state-of-art learners, supplied with an ad hoc knowledge. Rather, it allows to learn simple numerical relations without requirement for additional knowledge.

A second aspect of this work concerns the tractable characterization of the Disjunctive Version Space of consistent partially complete hypotheses. In opposition, as mentioned earlier, the theories built by either PROGOL or FOIL include only *a few* elements in this set.

Like PROGOL, ICP handles non ground examples, in opposition to FOIL [17]; but domain theory (that cannot be put as examples) can be considered only through saturation of the examples: ICP cannot use the domain knowledge in order to guide the exploration of the search space, as ML-Smart [1] or PROGOL do.

8.2 Generalization from constraints

As far as we know, the generalization from constraints has only been addressed so far by Page and Frisch [16] and Mizoguchi and Ohwada [12].

In [16], the goal is to generalize constrained atoms. Constrained atoms are handled as definite clauses whose antecedents express the constraints. Constrained generalizations of two atoms are built from the sorted generalizations defined on their arguments. In both [16] and our approach, generalization ultimately proceeds by building constraints. But different issues are addressed. In [16], the main difficulty arises from the possibly multiple generalizations of two

terms, which does not occur in our restricted language (section 4.2). In opposition, the main difficulty here comes from the multiple structural matchings among examples (section 7) while such a matching uniquely follows from the unique atom considered in [16].

Another approach of the generalization of constrained clauses is presented by Mizoguchi and Ohwada [12]. This work is nicely motivated by geometrical applications (avoiding the collision between objects and obstacles). The region of safe moves of an object can be 'naturally' described through a set of linear constraints; the goal consists in automatically acquiring such constraints from examples.
[12] first extend the definition of some typical induction operators (minimal generalization, absorption, lgg) to constrained clauses. Then, an *ad hoc* domain theory being given, examples are described by constrained atoms which are generalized through absorption and lgg, in the line of [15].

In what regards the roles respectively devoted to ILP and CLP, the essential differences can be summarized as follows: the induction of constrained clauses is done (a) by incorporating the structure of constraints into ILP, in [16]; (b) by extending the inverse resolution approach to CLP in [12]; and by interleaving ILP and CLP in our approach.

8.3 Reformulation

A strong motivation for reformulating ILP problems into simpler problems, e.g. in propositional form, is that propositional learners are good at dealing with numbers [8, 2, 26]. LINUS [8] achieves such transformation under several assumptions, which altogether ensure that one first-order example is transformed into *one* attribute-value example; this transformation thereby does not address the case of multiple structural matchings among examples. LINUS nicely uses the theory of the domain in order to introduce new variables and enrich the attribute-value representation of the examples.

Another approach is that of Zucker and Ganascia [26, 27], that focuses on restricting the set of predicates and substitutions relevant to a given level of induction. Simply put, moriological reformulations rely on a hierarchical description of the problem domain, where a morion of a given level can be decomposed into one or several morions of a lower level (e.g. the *car* morion involves the description of four *tire* morions). One may then restrict oneself to consider pattern matchings among examples, that preserve the structure (front tires, back tires). Such restrictions allow to drastically decrease the complexity of induction (which could benefit to ICP too); but the machine learning of such restrictions is still an open problem [26].

Note that [8] and [26] both map an induction problem into another simpler induction problem. In opposition, the mapping presented here enables a shift of paradigm: an induction problem is transformed into a constraint program, which can in turn be solved by an external tool.

8.4 Perspectives

This work opens several perspectives of research:

New variables (as in [8]) and new types of constraints could be considered. Ideally, language bias would be expressed via additional constraints (for instance, requiring the solution clauses to be connected could be expressed via additional constraints).

Also, the user could supply some optimality function in order to guide the selection of the admissible solutions. Selective discriminant induction could then be reformulated as a constrained optimization problem (finding the optimum of the objective function still satisfying the constraints).

But many promising tracks are opened by current experimental validations of this scheme [22].

Acknowledgments

This work has been partially supported by the ESPRIT BRA 6020 Inductive Logic Programming and by the ESPRIT LTR 20237 ILP^2.

References

1. F. Bergadano and A. Giordana. Guiding induction with domain theories. In Y. Kodratoff and R.S. Michalski, editors, *Machine Learning : an artificial intelligence approach*, volume 3, pages 474–492. Morgan Kaufmann, 1990.
2. S. Dzeroski, L. Todorovski, and T. Urbancic. Handling real numbers in ILP: a step towards better behavioral clones. In N. Lavrac and S. Wrobel, editors, *Proceedings of ECML-95, European Conference on Machine Learning*, pages 283–286. Springer Verlag, 1995.
3. A. Giordana, L. Saitta, and F. Zini. Learning disjunctive concepts by means of genetic algorithms. In Cohen W. and Hirsh H., editors, *Proceedings of ICML-94, International Conference on Machine Learning*, pages 96–104. Morgan Kaufmann, 1994.
4. ILOG. *Manuel SOLVER.* ILOG, Gentilly, France, 1995.
5. J. Jaffar and J. L. Lassez. Constraint logic programming. In *Proc. of the fourteenth ACM Symposium on the Principles of Programming Languages*, pages 111–119, 1987.
6. J. Jaffar and M.J. Maher. Constraint logic programming : a survey. *Journal of Logic Programming*, pages 503–581, 1994.
7. R.D. King, A. Srinivasan, and M.J.E. Sternberg. Relating chemical activity to structure: an examination of ILP successes. *New Gen. Comput.*, 13, 1995.
8. N. Lavrač and S. Džeroski. *Inductive Logic Programming: Techniques and Applications.* Ellis Horwood, 1994.
9. J.W. Lloyd. *Foundations of Logic Programming, second extended edition.* Springer Verlag, 1987.
10. R.S. Michalski. A theory and methodology of inductive learning. In R.S Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning : an artificial intelligence approach*, volume 1, pages 83–134. Morgan Kaufmann, 1983.

11. T.M. Mitchell. Generalization as search. *Artificial Intelligence*, 18:203–226, 1982.
12. F. Mizoguchi and H. Ohwada. Constraint-directed generalizations for learning spatial relations. In *Proceedings of ILP-91, International Workshop on Inductive Logic Programming*, 1991.
13. S. Muggleton. Inverse entailment and PROGOL. *New Gen. Comput.*, 13:245–286, 1995.
14. S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19:629–679, 1994.
15. S. Muggleton and C. Feng. Efficient induction of logic programs. In *Proceedings of the 1st conference on algorithmic learning theory*. Ohmsha, Tokyo, Japan, 1990.
16. C. D. Page and A. M. Frisch. Generalization and learnability: A study of constrained atoms. In S. Muggleton, editor, *Proceedings of the first International Workshop on Inductive Logic Programming*, pages 29–61, 1991.
17. J.R. Quinlan. Learning logical definition from relations. *Machine Learning*, 5:239–266, 1990.
18. M. Sebag. A constraint-based induction algorithm in FOL. In W. Cohen and H. Hirsh, editors, *Proceedings of ICML-94, International Conference on Machine Learning*, pages 275–283. Morgan Kaufmann, July 1994.
19. M. Sebag. Delaying the choice of bias: A disjunctive version space approach. In L. Saitta, editor, *Proceedings of the 13^{th} International Conference on Machine Learning*, pages 444–452. Morgan Kaufmann, 1996.
20. M. Sebag and C. Rouveirol. Induction of maximally general clauses compatible with integrity constraints. In S. Wrobel, editor, *Proceedings of ILP-94, International Workshop on Inductive Logic Programming*, 1994.
21. M. Sebag and C. Rouveirol. Constraint inductive logic programming. In L. de Raedt, editor, *Advances in ILP*, pages 277–294. IOS Press, 1996.
22. M. Sebag and C. Rouveirol. Tractable induction and classification in FOL. In *Proceedings of IJCAI-97*. Morgan Kaufmann, 1997.
23. A. Srinivasan and S. Muggleton. Comparing the use of background knowledge by two ILP systems. In L. de Raedt, editor, *Proceedings of ILP-95*. Katholieke Universiteit Leuven, 1995.
24. P. Van Hentenryck and Deville Y. Constraint Logic Programming. In *Proceedings of POPL'97*, 1987.
25. P. Van Hentenryck and Deville Y. Operational semantics of constraint logic programming over finite domains. In *Proceedings of PLILP'91*, 1991.
26. J.-D. Zucker and J.-G. Ganascia. Selective reformulation of examples in concept learning. In W. Cohen and H. Hirsh, editors, *Proc. of 11^{th} International Conference on Machine Learning*, pages 352–360. Morgan Kaufmann, 1994.
27. J.-D. Zucker and J.-G. Ganascia. Representation changes for efficient learning in structural domains. In L. Saitta, editor, *Proceedings of the 13^{th} International Conference on Machine Learning*, pages 543–551, 1996.